10g rapport

Henrik Flindt

Nicolas Dyhrman

Adrian Joensen

January 2, 2019

Wolf in Moose's Clothing

"A single type of bird can be called a goose, but more is geese, but the plural of moose is not meese, and finally one wolf becomes several wolves. English is like coding. Nobody really know why we do it that way we do, but people on the internet will yell at you for getting it wrong."
-Based on common saying.

The main assignment is the creation of a game, that mimic the relations between mooses and wolfs in a closed park. Each iteration of the game will see the animals move, reproduce and potentially eat a moose. The game is played from the command line interface, by calling its main function called `experimentWAnimals.exe` and give it eight arguments, that fits with the assignment specifications. If done correct, afther the program has run its cause a text file is created with information about the game.

1 Design

The overall sourcecode lay out is based upon the one given as part of the assignment, except the environment. To that the behaviors and events were added.

1.1 Animals

To move the animals, a random vector was pick based upon a random number from 0 to 7 (There are a total of 3x3-1 = 8 directions), where the direction was check if it's a valid, and if so change the animals' position with the vector.
To make the order of animals random, first the wolf list is selected and a tuole list is create a tuple list, where the first element will contain their symbol "w" and the second element will contain the index of the wolves. This is also done to the moose list. This is done for the entirety of the two lists, which are then joined and shuffled.
To shuffle the tuple list, a function with a for loop was used. It picks a random

index, remove the element from the tuple list and put it in a new list. When the tuple is empty, the new list is returned.. After this shuffle, this tuple is used to deside the order of succession.

1.1.1 Wolf

The wolves ability to eat a moose was implemented specifically for them, and works in a similarly as the move function, but with mooses being a favored direction compared to a empty or wolf occupied location. If the wolf sees one or more mooses next to it, it will pick one at random and move there. The moose is noted as dead and removed later.

1.2 Two-lists makes a board

2 Implementation

2.1 Manual

To compile the main function, type:
`fsharpc animalsSmall.fs experimentWAnimals.fsx`
To run the main function, type:
`mono experimentWAnimals.exe tic file.txt size moose fMoose wolf fWolf hWolf`
`tic` is the integer amount of iterations the game will run. `file.txt` is the name of the file where the data shall be saved. Must end with `.txt`. `size` is the size of the squared board. `moose` and `wolf` is the amount of the two animals respectively. `fmoose` and `fWolf` is the time interval in tics, of where the two different animals shall multiply. Finally `hWolf` is when a wolf ought to eat to avoid starvation and death.

```
/// An animal is a base class. It has a position and
    a reproduction counter.
type animal (symb : symbol, repLen : int) =
  let mutable _reproduction = rnd.Next(1,repLen)
  let mutable _pos : position option = None
  let _symbol : symbol = symb

  member this.symbol = _symbol
  member this.position
    with get () = _pos
    and set aPos = _pos <- aPos
  member this.reproduction = _reproduction
  member this.updateReproduction () =
    _reproduction <- _reproduction - 1
  member this.resetReproduction () =
    _reproduction <- repLen
```

```
override this.ToString () =
  string this.symbol
```

3 White Box Testing

The general move functions was visually tested on single animals setup, with a low reproduction and hunger rate. This is due to the random nature of the movements, and the overall visual aspect of the movement. It was discovered that several animals, that could reproduce, sometimes appeared not to move. This is believed to be a false negative test, caused by the animals moving at random in synchronization fo a single iteration. Therefor the test was done with a higher level of manual control.

Input	Expected	Result

4 Conclusion

While the program functions as expected, it does increase a disproportionate amount when the animal count or board size increases. This is believed to be cause by the high amount of lists being used to keep track of the animals. Each time an animal makes a move it checks the entire list as to ensure no animal is next to it. This ramification due to the design, was to monumental to redesign when discovered, but should be fixed in the case of more widespread use.